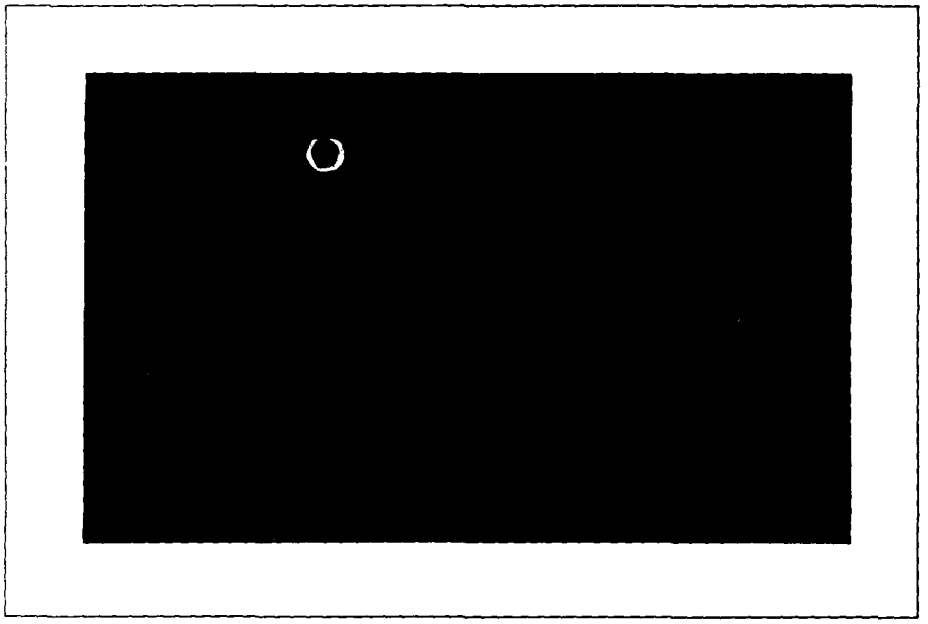
(2)



The Artificial Intelligence and Psychology Project

Departments of
Computer Science and Psychology
Carnegie Mellon University

Learning Research and Development Center
University of Pittsburgh

(2)

A TASK-ANALYTIC APPROACH TO THE AUTOMATED DESIGN OF INFORMATION GRAPHICS

Technical Report AIP - 82

Stephen Casner

Learning Research and Development Center
University of Pittsburgh
Pittsburgh, PA 15260

1989

This research was supported by the Computer Sciences Division, Office of Naval Research, under contract number N00014-86-K-0678. Reproduction in whole or part is permitted for any purpose of the United States Government. Approved for public release; distribution unlimited.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; Distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AIP - 82	5. MONITORING ORGANIZATION REPORT NUMBER(S) Same as Performing Organization
6a. NAME OF PERFORMING ORGANIZATION Carnegie Mellon University 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Personnel and Training Research Office of Naval Research (Code 1142PT)
6c. ADDRESS (City, State, and ZIP Code) Department of Psychology Pittsburgh, Pennsylvania 15213	7b. ADDRESS (City, State, and ZIP Code) 800 N. Quincy Street Arlington, VA 22217-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Same as Monitoring Organization 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-88-K-0086
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. N/A; PROJECT NO. N/A; TASK NO. N/A; WORK UNIT ACCESSION NO N/A

11. TITLE (Include Security Classification)

A task-analytic approach to the automated design of information graphics

12. PERSONAL AUTHOR(S) Stephen Casner

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM 86Sept15 TO 91Sept14	1989	42

16. SUPPLEMENTARY NOTATION

Submitted to ACM Transactions on Graphics

17 COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

graphic design; visual languages; task analysis; user interface; perception

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

SEE REVERSE SIDE

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION
22a. NAME OF RESPONSIBLE INDIVIDUAL Susan Chipman	22b. TELEPHONE (Include Area Code) (202) 696-4322 22c. OFFICE SYMBOL 1142 PT

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE

Unclassified

ABSTRACT

Graphical representations popularly thought to be useful for communicating and processing information yield mixed results when tested with real users. Cognitive research suggests that graphic design methodologies that focus primarily on the information to be presented in a graphic fail to exploit the potentials of graphics for expediting human performance of information-processing tasks: (a) allowing users to substitute efficient visual operators in place of more demanding logical operators; and (b) streamlining users' search for needed information. BOZ is a graphic design and presentation tool that constructively applies task-analytic principles of the efficiencies that graphics can offer to the problem of designing novel graphics that help streamline user tasks. BOZ analyses a procedural description of a user task and derives a provably equivalent visual task by substituting visual operators in place of logical operators. BOZ automatically designs and renders an accompanying graphic, encoding data in the graphic such that performance of each visual operator is supported and visual search is minimized. Graphics produced by BOZ are static 2D images that support interactive manipulations of the graphical objects in a display to allow direct modification of the internally stored information that the graphic depicts. BOZ is used to design a graphical alternative to a standard tabular display of airline schedule information to support an airline reservation task. Reaction time studies done with real users show that the BOZ-designed display significantly reduces users' performance time to the task. Regression analyses link the observed efficiency savings to BOZ's two key design principles: visual operator substitutions and pruning of visual search.

Keywords: Graphic Design, Task Analysis, Presentation, Visual Languages, User Interface, Military Applications, Procedures. (KR)

A Task-Analytic Approach to the Automated Design of Information Graphics

STEPHEN CASNER
University of Pittsburgh

Graphical representations popularly thought to be useful for communicating and processing information yield mixed results when tested with real users. Cognitive research suggests that graphic design methodologies that focus primarily on the information to be presented in a graphic fail to exploit the potentials of graphics for expediting human performance of information-processing tasks: (a) allowing users to substitute efficient visual operators in place of more demanding logical operators; and (b) streamlining users' search for needed information. BOZ is a graphic design and presentation tool that constructively applies task-analytic principles of the efficiencies that graphics can offer to the problem of designing novel graphics that help streamline user tasks. BOZ analyzes a procedural description of a user task and derives a provably equivalent visual task by substituting visual operators in place of logical operators. BOZ automatically designs and renders an accompanying graphic, encoding data in the graphic such that performance of each visual operator is supported and visual search is minimized. Graphics produced by BOZ are static 2D images that support interactive manipulations of the graphical objects in a display to allow direct modification of the internally stored information that the graphic depicts. BOZ is used to design a graphical alternative to a standard tabular display of airline schedule information to support an airline reservation task. Reaction time studies done with real users show that the BOZ-designed display significantly reduces users' performance time to the task. Regression analyses link the observed efficiency savings to BOZ's two key design principles: visual operator substitutions and pruning of visual search.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques--user interfaces; H.1.2 [**Models and Principles**]: User/Machine Systems--human information processing; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems; I.3.6 [**Computer Graphics**]: Methodology and Techniques--ergonomics.

General Terms: Design, Human Factors, Algorithms, Theory.

Additional Key Words and Phrases: graphic design, task analysis, perception, visual languages, user interface.

1. THE COGNITIVE FUNCTION OF GRAPHICAL DISPLAYS

A striking conclusion of recent studies concerned with understanding how and why graphical representations are useful is that it is a false assumption that graphical displays are inherently better than other representations, or that perceptual inferences are always made

more efficiently than non-perceptual inferences [18, 23]. Rather, these studies suggest that the usefulness of a graphic is a function of the *task* that the graphic is being used to support. Twenty-nine independent empirical studies surveyed in Jarvenpaa and Dickson [18] found graphical displays superior to tabular displays for a restricted set of information-processing tasks, and observed no benefits or poorer performance for other tasks. Examples of graphics that succeed in practice are best explained as "situationally dependent artifacts [18]" whose success arises out of the combination of task performed and the particular graphic used. Generalizations made about the observed usefulness of a graphic for one task are highly inappropriate since using the same graphic for different tasks typically cause the usefulness of the graphic to disappear. Consequently, graphic design principles such as "line graphs are best for continuous data" are too underspecified to be useful in general. That is, empirical studies have shown that line graphs are supportive of some tasks that manipulate continuous data and are detrimental to the performance of others. The implication is that effective graphic design should begin with the task that a graphic is intended to support, and be focused on finding those parts of a task, if any, that might be performed more efficiently within the context of a graphical display.

Larkin and Simon's [23] theoretical analysis points out two ways in which graphical displays can expedite human performance of information-processing tasks:

- **Substituting visual operators for logical operators**: Graphical displays often allow users to substitute quick and easy perceptual judgements (visual operators) in place of more demanding non-visual reasoning steps (logical operators) that comprise a task. Visual operators such as distance and size determinations, spatial coincidence judgements, and color comparisons, sometimes give users the same information as more demanding logical operators such as mental arithmetic, logical reasoning steps, or feature comparisons.

- **Reducing search**: Good graphics often reduce the time that the user must spend searching for information they need. This is accomplished either by grouping together information required to draw a particular inference into one spatial locality, or by employing techniques such as shading and spatial arrangement that help guide the eye toward relevant information and away from irrelevant information.

To illustrate the two ways that graphical displays can help users, Figure 1 shows a graphic used for train schedules in France in the late 1800's [26].

****
Figure 1 here
****

Marey's train schedule is a 2-dimensional visual data structure that indexes time and place information along the horizontal and vertical axes, respectively. To retrieve departure and arrival times for a train, the user must perform coincidence judgements along the horizontal axis. Note that for this simple task in isolation, the train schedule does not result in any savings for the user. Searching for departure and arrival times in a tabular presentation such as Figure 2 would seem to progress as quickly and perhaps more accurately. Consequently, with respect to the task of retrieving departure and arrival times, allowing users to substitute spatial coincidence judgements for table lookup seems to be of no use in that it may be both inefficient to use and prone to errors due to the imprecise representation of times. The empirical studies generalize this to show that in most cases tabular representations are best for "information extraction" tasks [18], and thus there seem to be no inherent advantages of representing information graphically.

****
Figure 2 here
****

When used for other tasks, the graphical train schedule offers several advantages by allowing the user to substitute visual operators in place of more difficult logical operators that would ordinarily require logical reasoning and mental arithmetic, and reducing the time the user must spend searching for information. For example, to find a route between Paris and Lyon we can search for a single line that runs directly from Paris to Lyon, or find a series of lines such that each successive line lies to the right of the previous line. The savings in search time is achieved because of the way the graphical display indexes the trains by city and time. Notice that accomplishing this same task with the tabular display requires that we continually search the entire list of trains since they are not indexed by city. We can also determine the speed of a train by judging the slope of the line between cities. Rather than dividing the total number of miles traveled by the difference of the departure and arrival times, we can compare the speeds of trains by performing simple slope judgements. We can determine the layover between two trains in an intermediate city by estimating the distance between the end of the line depicting the first train and the beginning

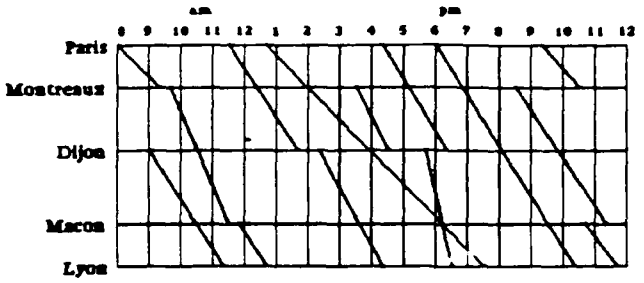


Figure 1: Marey's Train Schedule

Origin/Destination	Departs	Arrives
Paris-Montreaux	8:00	9:15
Dijon-Lyon	9:00	11:15
Montreaux-Macon	9:45	11:30
Paris-Dijon	11:30	1:30
Macon-Lyon	11:45	12:45
Paris-Lyon	12:30	6:30
Dijon-Lyon	2:30	4:15
Montreaux-Dijon	3:30	4:30
Paris-Dijon	4:30	6:15
Dijon-Lyon	5:30	6:30
Paris-Lyon	6:00	10:15
Montreaux-Macon	8:30	11:30
Paris-Montreaux	9:30	10:30
Macon-Lyon	10:45	11:45

Figure 2: A Tabular Train Schedule

of the line that depicts the second train. This convention allows the user to substitute a simple distance judgement in place of subtracting the departure and arrival times.

Note that a different user who wishes to understand the route structure of French trains would find both of the displays shown in Figures 1 and 2 cumbersome and would benefit most from the graphic shown in Figure 3. This structuring of the data trades away the departure/arrival time capabilities to allow users to find a city quickly (indexed by geographical location), and to determine routes by performing connectivity judgements between the city names.

****
Figure 3 here
****

In summary, the three alternative ways of presenting the same information show that different graphics best support different tasks. Consequently, effective graphical displays are not likely to follow from a design methodology concerned primarily with the information to be displayed but rather from a careful analysis of the tasks that manipulate the information.

Overview. The research described in this paper explores an approach to the design of information graphics based on an analysis of the tasks for which they are intended to support. The design approach is implemented in an automated graphic design and presentation tool called BOZ. The core idea behind BOZ can be summarized as follows: *since the potential advantages of graphics are task-related, graphic design activities should focus on designing efficient visual tasks. Decisions made about how to encode and structure information in an accompanying graphic should be based primarily on supporting efficient and accurate performance of the visual task.* The enabling step in the task-analytic approach is to capture the notion of a visual task performed by human users using the same formal framework used to describe arbitrary computational processes, allowing design decisions to follow formal criteria.

Section 2 reviews previous work related to the problem of designing information graphics. Section 3 overviews the four main components of BOZ and introduces a running example used throught the paper to describe BOZ's approach. Sections 4 through 7 describe in detail the four components of BOZ's design methodology. Section 4 shows how sets of alternative visual procedures can be derived from a logical procedure by substituting visual

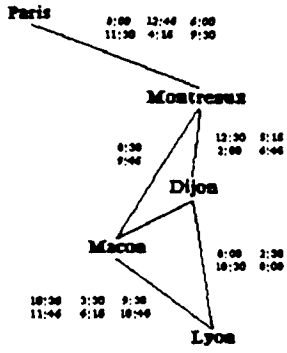


Figure 3: An Alternative Graphical Train Schedule
That Supports a Different Task

operators in place of logical operators when the logical and visual operators can be shown to yield the same result. Section 5 shows how an analysis of the relationships between task operators can be used to determine how information can be structured within a graphic such that visual search is minimized when users perform a visual procedure. Section 6 describes three criteria used by BOZ when selecting a single visual procedure and graphic design that best supports performance of the user's task. Section 7 describes an automated rendition component that graphically renders arbitrary sets of relational facts using the prescribed graphic design. Section 8 uses Larkin and Simon's theoretical criteria to analyze a visual procedure and graphic designed by BOZ. The analysis produces a set of specific hypotheses about the potential task performance efficiencies of the BOZ-designed procedure and graphic. Finally, Section 9 reviews an experiment reported in Casner and Larkin [7] in which participants performed the visual procedure using a series of accompanying experimental graphics. The experimental graphics were designed to explicate the contribution made by each efficiency advantage hypothesized. Results show significant decreases in users' performance times and suggest that users obtained the efficiency savings through the hypothesized visual operator substitutions and visual search reductions.

2. PREVIOUS WORK

The following surveys theoretical and empirical work concerned with the problem of designing effective graphics.

2.1 Graphic Design Practices

Tufte [32], Bertin [2], Cleveland [10], and Schmid [28] describe practices for designing graphical displays for the purpose of communicating information. These practices help designers to make use of graphic techniques that have been observed to help users, and to avoid bad practices known to make graphics ambiguous, confusing, or generally less usable. There are three basic limitations of the design practices approach. First, taken together, the graphic design practices do not comprise a top-down, prescriptive theory of how to begin designing a graphic from scratch. Second, the graphic design practices focus mainly on the information to be presented in a graphic and include less concern for the tasks for which the graphics are designed to support. Third, the graphic design practices are concerned mainly with graphics used for the purpose of communicating information to the user. This focus overlooks the potential of graphics as tools for supporting the performance of complex information-processing tasks.

2.2 Automated Graphical Presentation Systems

APT [25] is an automated graphic presentation tool that designs non-interactive graphical presentations of relational information. A significant contribution of APT was to formally characterize something that many previous investigators informally alluded to: that graphical presentations can be expressed as sentences in a formal graphical language that have the same precise syntax and semantics as propositional formalisms. The advantage of having a formalism for graphical presentations is that it provides a set of criteria for deciding the role of each visible sign or symbol placed in a graphic, and improves the integrity of a graphical display by using formal methods for transforming relational facts to visual facts. APT's style of analysis for formal graphical languages has been used by nearly every graphics presentation tool designed after APT, including the one described in this paper. A second contribution of APT is that, unlike proposals for graphic design practices, APT designs graphics with a minimum amount of intervention on the part of the designer. That is, APT embodies a genuinely *prescriptive* theory of how to design a graphic. However, APT's design algorithm is based entirely on an analysis of the information to be presented and does not consider the task for which a graphic is to be used. This prevents APT from directly exploiting the task-related advantages of graphics, and from creating different displays of the same information to support different tasks.

SAGE [27] is a hybrid, text and graphics presentation system that generates explanations of changes that occur in quantitative modeling systems such as project modeling systems and financial spreadsheets. Graphical presentations are designed by SAGE in response to information queries made by the user. Through an analysis of user queries, SAGE's design of graphical presentations is sensitive to the goals of the user, taking an important step toward exploiting the task-related advantages of graphics. SAGE presently contains only a small set of primitive problem-solving operators and is not able to fashion displays to support complex information-processing tasks involving combinations of many primitive operators. SAGE does not make use of any theory of display-based problem solving when designing a display to support a user's task.

AIPS [36] accepts descriptions of information encoded using the KL-ONE [3] knowledge representation language. AIPS matches the KL-ONE descriptions against a set of pre-defined display formats and chooses that format that best matches the characteristics of the data. AIPS is not able to design new graphical displays.

BHARAT [15] accepts descriptions of data sets and chooses one of bar chart, pie chart, and line chart to present relations between the data. The display format chosen is determined by the characteristics of the data: line charts are used for continuous data, pie charts for proportional data, and bar charts for all others. Like AIPS, BHARAT chooses among existing displays rather than designing new ones.

VIEW [14] creates graphical presentations of information about ships maintained in a naval database. VIEW's knowledge base contains information about particular users, a set of tasks for the domain, and a set of pre-defined KL-ONE descriptions of possible presentation formats. By matching users' identities, tasks, and queries against the presentation format descriptions, VIEW is able to present different graphics in different contexts. As with AIPS and BHARAT, VIEW does not design its presentation formats.

APEX [13] creates graphical explanations of actions performed with physical devices in a 3-dimensional world. Explanations are created by presenting sequences of static images depicting the individual steps in an action. APEX uses hierarchical descriptions of the objects that can appear in an explanation, where each level in the hierarchy contains more detailed features of the object. A second mechanism allows APEX to determine how much detail is needed at each step and to display only that information. Since the objects that appear in explanations are highly domain-specific, they must be hand-created prior to using APEX.

2.3 Cognitive Research on the Utility of Graphics

Larkin and Simon's work [23] was first to study the utility of graphics from a cognitive science perspective. Larkin and Simon built detailed cognitive simulations of human task performance, each simulation performing a task using informationally equivalent logical and graphical representations of a set of data. Larkin and Simon's analysis yielded two ways in which graphical display-based procedures could be performed more efficiently by humans: (1) by allowing users to substitute quick visual operators for more demanding logical operators; and (2) by reducing search for needed information.

Several more recent studies have investigated other cognitive utilities of graphical displays. Hegarty and Just [16] studied the use of diagrams in understanding complex machines and showed diagrams to be effective in reducing the amount of information that must be maintained in short-term memory. Fallside [12] investigated how learners make use of animated explanations similar to those produced by Feiner's APEX program when

understanding complex machines. Koedinger and Anderson's work [22] suggests that students are better able to learn and apply the rules of geometry when the rules are explained to them in terms of the way they can be applied to particular configurations of lines and angles in geometry diagrams.

In short, cognitive research has served to replace intuitive notions of why graphics are useful with a more scientific understanding of how graphics leverage human performance of information-processing tasks. A new generation of graphic design techniques is now possible that focuses on studying users' tasks and determining ways to support those tasks in the context of a visual display. The research described below is the first to approach the graphics design problem from a task perspective, placing less emphasis on analyses of the information to be presented in a graphic.

3. BOZ: TASK-ANALYTIC DESIGN OF INTERACTIVE INFORMATION GRAPHICS

The following describes a graphic design methodology used to create interactive graphical presentations of relational information based on an analysis of the tasks that a graphic is intended to support. The graphic design theory is articulated in an automated graphic design and presentation tool called BOZ. BOZ's design activities focus on designing visual procedures, to be performed by human users, that allow users to accomplish a stated goal more efficiently than they would be able to without the benefit of a graphical display. BOZ's task-analytic approach uses the following four components.

A **visual operator substitution** component analyzes descriptions of logical procedures, substituting visual operators in place of logical operators when the operators can be shown to produce the same output given the same input. Visual operator substitution is the mechanism used to reduce the amount of mental computation performed by the human user when performing a task. Several visual operators typically qualify as substitutes for each logical operator, yielding a set of possible visual procedures.

A **visual data structuring** component examines the information manipulated by each logical operator and determines how information shared by several operators should be collected together in the same spatial locality. Visual data structuring is one mechanism used to minimize the amount of time the user spends searching for information in a graphic. The visual data structuring component indicates how information is to be grouped within a display or partitioned among distinct displays but does not determine how the information is to be visually encoded in the graphic.

A **visual operator selection** component chooses a single visual operator from the list of possible visual operators produced by the visual operator substitution component. The first criteria for operator selection is how efficiently and accurately each visual operator is likely to be performed by human users. Selecting each particular visual operator also decides the way that the information manipulated by that operator (and related operators) must be visually encoded in a graphic. A second criteria for operator selection is choosing a complete set of visual operators that results in a set of graphical encodings that can be combined according to the specification produced by the visual data structuring component. The results of applying the visual operator selection component are detailed descriptions of a single visual procedure and an accompanying graphic design that supports the performance of the visual procedure. Together, the visual data structuring and operator selection components allow BOZ to visually organize and present information in novel ways, customized to a visual task, rather that choosing from a set of "canned" graphic designs.

A **rendering** component translates relational facts into visual facts and displays them on the computer screen in the format specified by the visual data structure description. Graphics produced by the rendering component support two-way interactions that allow changes made in the internally-stored relational facts to be automatically reflected in the graphical display, and direct manipulations of the graphical objects in the display to be reflected in the internal set of relational facts.

The next four sections describe BOZ's four components in detail. To illustrate how BOZ works, a running example will be developed throughout the discussion. In the example, a visual display will be designed to support the following task pertaining to making a reservation on an airline flight:

> *Find a pair of connecting flights that travel from Pittsburgh to Mexico City. You are free to choose any intermediate city as long as the layover in that city is no more than four hours. Both flights that you choose must be available. The combined cost of the flights cannot exceed $500.*

Figure 4 shows a computer flight information display obtained from a local travel agency. Travel agents use the display in Figure 4 to assist them in processing customers requests for flights. The purpose of the running example used throughout the paper is to show how

BOZ can be used to analyze a real-world task and design a visual procedure and accompanying display to help users accomplish the same task more efficiently.

****
Figure 4 here
****

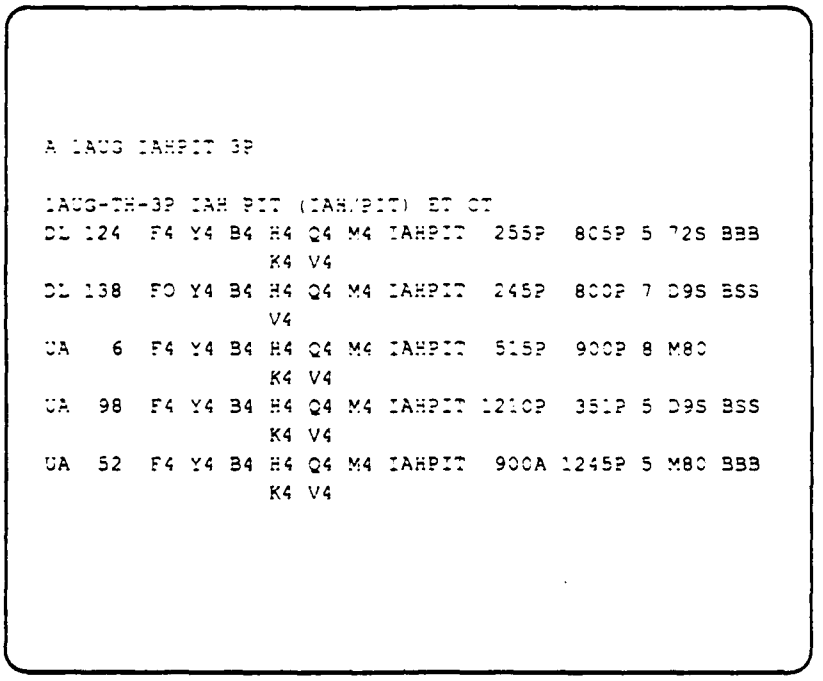
4. VISUAL OPERATOR SUBSTITUTION

The visual operator substitution component analyzes descriptions of logical procedures substituting perceptual operators in place of logical operators. Visual operator substitution is the graphic design technique used to insure that a graphical display best exploits the first type of cognitive advantage of graphics: that users can substitute efficiently performed perceptual inferences in place of more demanding logical inferences. The visual operator substitution component produces a set of visual operators that can potentially serve as substitutes for each logical operator. Decisions about which particular visual operator to substitute for each logical operator are subject to further design criteria described in Section 6.

Visual operator substitution relies on three important components. A *logical task description language* is used to enumerate the logical operators necessary to complete a task, and a set of relational facts that are manipulated by the operators. A *catalog of visual operators* describes information-processing activities that occur within the context of a visual display. A *substitution algorithm* considers each logical operator in a task and searches the catalog of visual operators for those visual operators that compute the same function as the logical operator. Since there are often several perceptual operators that qualify as substitutes for a logical operator, the visual procedure derivation component produces a set of possible visual procedures that accomplish the logical task.

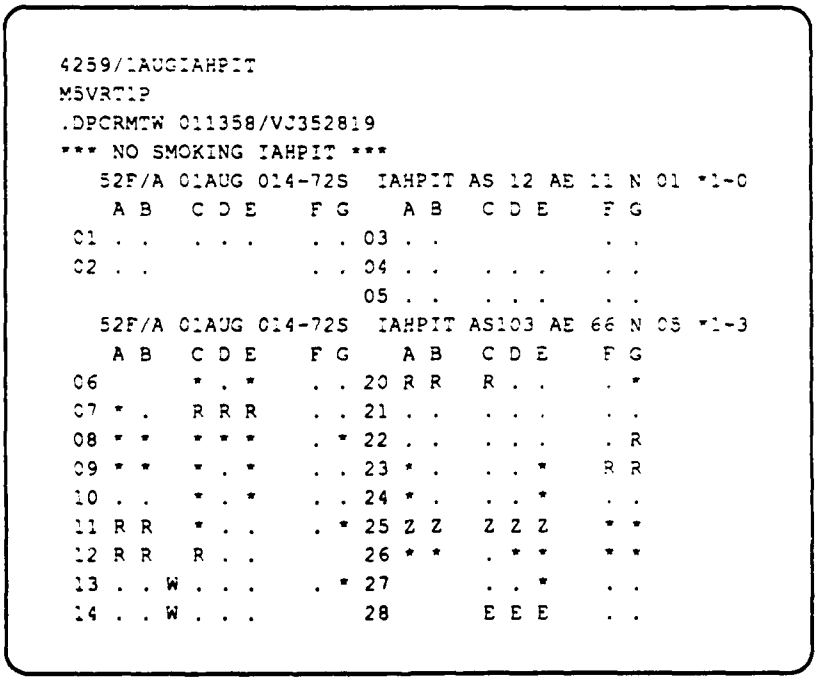
4.1 Logical Procedure Description Language

The first component of any task-oriented design methodology is a means of making explicit the information-processing activities that a display is intended to support. The following describes the language used to describe tasks performed by the user. These descriptions must presently be generated by hand and submitted to BOZ as input. Section 10 discusses prospects for the machine generation of task descriptions. The language contains two basic components: (a) a notation for describing logical procedures; and (b) a notation for expressing relational facts manipulated by a logical procedure.



```
A IAUG IAHPIT 3P

IAUG-TH-3P IAH PIT (IAH/PIT) ET CT
DL 124  F4 Y4 B4 H4 Q4 M4 IAHPIT  255P  805P 5 72S BBB
                  K4 V4
DL 138  FO Y4 B4 H4 Q4 M4 IAHPIT  245P  800P 7 D9S BSS
                  V4
UA   6  F4 Y4 B4 H4 Q4 M4 IAHPIT  515P  900P 8 M80
                  K4 V4
UA  98  F4 Y4 B4 H4 Q4 M4 IAHPIT 1210P  351P 5 D9S BSS
                  K4 V4
UA  52  F4 Y4 B4 H4 Q4 M4 IAHPIT  900A 1245P 5 M80 BBB
                  K4 V4
```



```
4259/IAUGIAHPIT
M5VRT1P
.DPCRMTW 011358/VJ352819
*** NO SMOKING IAHPIT ***
   52F/A 01AUG 014-72S  IAHPIT AS 12 AE 11 N 01 *1-0
    A B   C D E    F G    A B   C D E    F G
 01 . .   . . .    . . 03 . .             . .
 02 . .             . . 04 . .   . . .    . .
                       05 . .   . . .    . .
   52F/A 01AUG 014-72S  IAHPIT AS103 AE 66 N 05 *1-3
    A B   C D E    F G    A B   C D E    F G
 06       * . *    . . 20 R R   R . .    . *
 07 * .   R R R    . . 21 . .   . . .    . .
 08 * *   * * *    . * 22 . .   . . .    . R
 09 * *   * . *    . . 23 * .   . . *    R R
 10 . .   * . *    . . 24 * .   . . *    . .
 11 R R   * . .    . * 25 Z Z   Z Z Z    * *
 12 R R   R . .        26 * *   . * *    * *
 13 . . W . . .    . * 27       . . *    . .
 14 . . W . . .        28       E E E    . .
```

Figure 4: Airline Schedule Display Obtained from a Local Travel Agency

Logical procedure definitions are similar to programs in conventional programming languages such as Pascal. Every logical procedure contains two parts: (a) a set of logical operator definitions; and (b) a main body. A *logical operator (LOP)* is composed of an operator name, a list of arguments taken as input to the operator, and a single relation that the operator computes. Logical operators occur in two forms. A *search operator* uses one of the three meta-commands: ASK, TELL, and RETRACT to query, assert, and remove relational facts from a simple database of relational facts. *Relational facts* contain a single predicate followed by any number of arguments. *Predicates* describe relations between two or more relational arguments. *Arguments* are variables that can be assigned relational values drawn from the collection of domain sets (explained below) defined for a task. In LOP definitions, arguments are indicated by the name of a domain set enclosed in brackets (i.e., "< >"). Arguments may be instantiated or uninstantiated. *Uninstantiated arguments*, i.e., variables that have not yet been assigned a value, are capitalized. *Instantiated arguments* are variables that were previously uninstantiated but have since been assigned a relational value. Instantiated arguments appear in lower case. Note that the same argument may be uninstantiated in one clause, be assigned a value, and appear in lower-case form (i.e., instantiated) in a later clause. A *computation operator* describes computations performed on a set of relational arguments using one of a set of pre-defined arithmetic or logical predicates such as PLUS, DIFFERENCE, AND, OR, NOT, etc. Note that only instantiated arguments may appear in a computation operator.

The following examples describe two logical operators in the airline reservation procedure. The two operators determine the departure time of an airline flight (search), and the layover between two flights (computation), respectively:

```
(NLAMBDA determine-departure (<flight> <DEPARTURE>)
  (ASK (Departure <flight> <DEPARTURE>)))

(NLAMBDA compute-layover (<departure> <arrival> <LAYOVER>)
  (DIFFERENCE <departure> <arrival> <LAYOVER>))
```

The keyword NLAMBDA is used to denote a logical operator. The lists (<flight> <DEPARTURE>) and (<departure> <arrival> <LAYOVER>) are the sets of arguments that the two operators receive as input. The ASK predicate states that a list of facts should be checked to see if the predicate that follows can be shown to be true: namely if there exist a fact expressing the departure time of the flight. The clause (DIFFERENCE <departure> <arrival>) specifies that the pre-defined subtraction predicate is to be

computed given the values <departure> and <arrival>, and the variable <LAYOVER> is to be instantiated with the result.

The *main body* of a logical procedure is an ordered sequence of calls to the set of defined logical operators. To express control information, the main body of a logical procedure may additionally contain any of the following control constructs: **while-do, for, repeat-until,** and **if-then.**

To illustrate how logical procedures are described using the task language, Figure 5 shows a complete procedure description for the airline reservation task.

```
(TASK airlineReservation

(DOMAINSETS
  (flight NOMINAL 50)
  (origin NOMINAL (pit hou dal ord alb mex gdl qto paz bga))
  (destination NOMINAL (pit hou dal ord alb mex gdl qto paz bga))
  (departure QUANTITATIVE 100)
  (arrival QUANTITATIVE 100)
  (layover QUANTITATIVE 100)
  (cost QUANTITATIVE 500)
  (availability NOMINAL boolean)
  (seats ORDINAL (1A 1B 1C 1D 1E 1F ... 24A 24B 24C 24D 24E 24F))

(LOPS
  (NLAMBDA findFlightWithOrigin (<FLIGHT> <origin>)
    (ASK (Origin <FLIGHT> <origin>)))
  (NLAMBDA findDestination (<flight> <DESTINATION>)
    (ASK (Destination <flight> <DESTINTATION>)))
  (NLAMBDA available? (<flight>)
    (ASK (Availability <flight>)))
  (NLAMBDA determineDeparture (<flight> <DEPARTURE>)
    (ASK (Departure <flight> <DEPARTURE>)))
  (NLAMBDA determineArrival (<flight> <ARRIVAL>)
    (ASK (Arrival <flight> <ARRIVAL>)))
  (NLAMBDA computeLayover (<departure> <arrival> <LAYOVER>)
    (DIFFERENCE <departure> <arrival> <LAYOVER>))
  (NLAMBDA layoverLessThanX? (<layover> <layover>)
    (LESSP <layover> <layover>))
  (NLAMBDA determineCost (<flight> <COST>)
    (ASK (Cost <flight> <COST>)))
  (NLAMBDA addCosts (<cost> <cost> <COST>)
    (PLUS <cost> <cost> <COST>))
  (NLAMBDA costLessThanX? (<cost> <cost>)
    (LESSP <cost> <cost>))
  (NLAMBDA findSeat (<flight> <SEAT>)
    (ASK (Seat <flight> <SEAT>)))
  (NLAMBDA emptySeat? (<seat>)
    (ASK (Availability <seat> T)))
```

```
(PROCEDURE
 (repeat
  (if (findFlightWithOrigin FLIGHT 'pit) then
   (if (available? flight) then
    (findDestination flight DESTINATION)
    (determineArrival flight ARRIVAL)
    (repeat
     (if (findFlightWithOrigin CONNECTING destination) then
      (if (available? connecting) then
       (determineDeparture connecting DEPARTURE)
       (computeLayover departure arrival LAYOVER)
       (if (layoverLessThan4? layover) then
        (determineCost flight COST1)
        (determineCost connecting COST2)
        (addCosts cost1 cost2 TOTAL)
        (if (lessThan500? total) then
         (repeat
          (findSeat flight SEAT1)
         (until (emptySeat? seat1)))
         (repeat
          (findSeat connecting SEAT2)
         (until (emptySeat? seat2)))))))
     (until done))))
 (until done))
```

Figure 5: Logical Airline Reservation Procedure

Relational facts are used to describe relational information manipulated by a logical procedure. Relational facts state relations between values drawn from one or more domain sets. *Domain sets* are the information types that define the universe of discourse for a task. Domain sets associate a name, a type of information, and a (possibly infinite) set of data values of that type that can appear in a relational fact. Three types of domain sets are allowed in the present model: *quantitative*, *nominal* and *ordinal* [30].

The airline reservation procedure manipulates information from nine domain sets which are specified in the top portion of Figure 5. Figure 6 shows a set of relational facts that describe a set of three airline flights.

```
(flight flight117)                    (cost flight117 179)
(flight flight738)                    (cost flight738 219)
(flight flight839)                    (cost flight839 319)
(origin flight117 pit)                (departure flight117 10:00)
(origin flight738 pit)                (departure flight738 8:00)
(origin flight839 pit)                (departure flight839 9:15)
(destination flight117 hou)           (arrival flight117 12:50)
(destination flight738 alb)           (arrival flight738 12:00)
(destination flight839 jfk)           (arrival flight839 12:05)
(availability flight117 ok)
(availability flight738 ok)
(availability flight839 ok)
```

Figure 6: Relational Facts for the Airline Reservation Task

4.2 A Catalog of Visual Operators

Visual operators, analogous to logical operators, characterize information-processing activities performed within the context of a visual display and whose performance depends on the use of a visual display. Visual operators occur in two forms. *Perceptual operators (POPs)* describe mental computation or visual search performed using graphically expressed information. For example, judging the distance between two objects in a display, and locating an object having a particular color are perceptual operators. *Graphical operators (GOPs)* describe interactive manipulations of the visual objects in a display. Moving an object from one location to another, and resizing an object to match the size of another object are examples of graphical operators.

Visual operators are organized around a set of *primitive graphical languages* available to the designer of a graphical display [25]. Primitive graphical languages comprise the designer's resources for encoding information graphically. The set of primitive graphical languages used in the present model are shown in Table I.

TABLE I
Primitive Graphical Languages

Horizontal Position	Color
Vertical Position	Labels
Cartesian Position	Line Thickness
Height	Line Dashing
Width	Slope
Line Length	Shape
Area	Visibility
Connectivity	Tabular
Shading	

Associated with each of the primitive graphical languages is a set of perceptual and graphical operators (POPs and GOPs) that are admitted when the designer of a graphical representation elects to use one or more of the primitive languages in a graphic. For example, if we elect to use the Horizontal Position language we admit a family of perceptual operators (POPs) such as determining the horizontal position of a graphical object, comparing two or more horizontal positions, and finding the midpoint of an interval defined by two horizontal positions. Horizontal position also admits a set of graphical operators (GOPs) such as moving a graphical object from one position to another. Table II shows the set of perceptual and graphical operators (POPs and GOPs) admitted by the Horizontal Position and Shading primitive graphical languages.

TABLE II
Perceptual Operators (POPs)

Horizontal Position	**Shading**
determine-horz-pos	determine-shade
search-object-at-horz-pos	search-object-with-shade
search-any-horz-pos-object	search-object-and-shade
verify-object-at-horz-pos	verify-object-and-shade
horz-coincidence?	darker?
left-of?	lighter?
right-of?	same-shade?
horz-forward-projection	find-lightest-shade
horz-backward-projection	find-darkest-shade
determine-horz-distance	shade-object
determine-horz-maximum	overlay-shaded-objects
determine-horz-minimum	
find-midpoint-of-horz-interval	
find-horz-sub-interval	
determine-interval	
same-size-horz-interval?	
smaller-horz-interval?	
bigger-horz-interval?	
left-of-group?	
right-of-group?	
group-lines-up?	
horz-move	
horz-overlay	
pop-up-at-horz-position	

Performance of a visual operator requires that the information manipulated by the visual operator is graphically encoded using the primitive graphical language that corresponds to the operator. For example, performing the `determine-horz-distance` operator requires that a graphic encode the information relevant to the operator using graphical objects meaningfully positioned along a horizontal axis.

Equivalence Classes for Visual Operators. Every visual operator computes a function over relational information. An *equivalence class* of visual operators is a set of operators that can be shown to compute the same function over relational information. Table III describes nineteen equivalence classes used to categorize the visual operators in the catalog. The relational function computed by each equivalence class is formally specified by an *operator schema.* The details of operator schemas are discussed below.

TABLE III
Operator Equivalence Classes

SEARCH OPERATORS

search: Search a list of facts for an object with a specified attribute value.
lookup: Search a list of facts about a particular object and report the value of a specified attribute of that object.
search and lookup: Search a list of facts for any object and report the value of a specified attribute of that object.
verify: Search for a fact stating that a specified object has a specified attribute value.

COMPUTATION OPERATORS

equal: Is one relational value equal to another?
lessthan: Is one ordinal or quantitative value less than another?
greaterthan: Is one ordinal or quantitative value greater than another?
plus: Computes the sum of two numbers.
difference: Computes the difference of two numbers.
times: Computes the product of two numbers.
quotient: Computes the quotient of two numbers.
setplus: Add corresponding elements in two lists of numbers.
setdifference: Subtract corresponding elements in two lists of numbers.
setequal: Apply the equality operator to corresponding elements in a list of relational values.
setlessthan: Apply the less than operator to corresponding elements in a list of relational values.
setgreaterthan: Apply the greater than operator to corresponding elements in a list of relational values.
max: Return the greatest value in a list of quantitative values.
min: Return the smallest value in a list of quantitative values.

GRAPHICAL OPERATORS

tell: Assert a new relational fact and retract any existing facts that contradict the new fact.

Each equivalence class contains visual operators drawn from the visual operator sets associated with each primitive graphical language. It is interesting to study the members of each equivalence class to see what types of operators each primitive graphical language supports. Interesting differences among languages may show why some techniques for encoding information in a graphic support certain reasoning steps more effectively than others. Table IV shows the visual operators associated with the **search** and **subtraction** equivalence classes.

TABLE IV
Members of Two Visual Operator Equivalence Classes

search	**subtraction**
search-object-at-horz-pos	determine-horz-distance
search-object-at-vert-pos	determine-vert-distance
search-object-at-cart-pos	determine-cart-distance
search-object-with-height	determine-height-difference
search-object-with-width	determine-width-difference
search-line-with-length	determine-difference-in-line-length
search-object-with-area	determine-area-difference
search-connected-object	subtract-labels
search-object-with-shading	determine-slope-difference
search-object-with-color	subtract-table-entries
search-object-with-label	
search-line-with-thickness	
search-line-with-dashing	
search-line-with-slope	
search-object-with-shape	
search-visible-object	
search-entry-in-table	

Visual operators are formalized using the same representation scheme used to describe tasks. For example, the `determine-horz-distance` operator is formalized as follows:

```
(NLAMBDA determine-horz-distance (<horzpos> <horzpos> <DISTANCE>)
  (DIFFERENCE <horzpos> <horzpos> <DISTANCE>))
```

4.3 Substituting Operators

The visual operator substitution algorithm analyzes logical procedures and attempts to locate visual operators that can be performed by human users more efficiently. This is accomplished by considering each logical operator (LOP) in a task description and searching the set of perceptual and graphical operators to locate those POPs and GOPs that can be shown to compute the same function as the LOP. This property insures that whatever visual procedure is followed in place of a corresponding logical procedure, it is guaranteed that the user will obtain the same results if the visual procedure is performed correctly.

Visual operators can qualify as substitutions for logical operators in two ways. *Simple substitutions* are those in which a single visual operator can be shown to be equivalent to a logical operator. *Complex substitutions* are those in which two or more visual operators can be packaged together using a rule for the composition of operators [4] to arrive at a complex visual operator that matches the logical operator.

Simple substitutions. A single visual operator qualifies as a substitute for a logical operator if and only if the logical operator can be categorized in the same equivalence class (see Table III) as the visual operator. Categorization is determined by attempting to match a LOP to the operator schemas used to describe each equivalence class. If a LOP is successfully matched to the schema of an operator class, all visual operators in that class initially qualify as simple substitutions for the LOP. For example, given the `compute-layover` logical operator in the airline reservation procedure:

```
(NLAMBDA compute-layover (<departure> <arrival> <LAYOVER>)
  (ASK (DIFFERENCE <departure> <arrival> <LAYOVER>)))
```

BOZ attempts to classify the LOP into each of the equivalence classes given in Table III until a class is found or the set of classes is exhausted. The `compute-layover` LOP can be successfully categorized into the **subtraction** class of search operators. One member of the **subtraction** class is the `determine-horz-distance` operator associated with the Horizontal Position language:

```
(NLAMBDA determine-horz-distance (<object> <object> <DISTANCE>)
  (ASK (DIFFERENCE <object> <object> <DISTANCE>)))
```

Since both operators compute the subtraction function, any graphic that represents departure and arrival times as objects positioned along a horizontal axis will always allow the user to perform the visual operator and obtain correct answers.

Complex substitutions. The substitution algorithm sometimes exhausts the list of equivalence classes without successfully categorizing a LOP. One situation in which this occurs is when a logical search operator contains a relation that takes more arguments than the relations contained in any single visual search operator. For example, suppose an operator queries a 3-place relation such as: "find the brother of Heather," or "find the sister of Alison." This task can be represented using the following single LOP:

```
(LAMBDA find-certain-relative-of-x? (<person> <PERSON> <relation>)
  (ASK (Related <person> <PERSON> <relation>)))
```

If we consider the schemas for each of the equivalence classes in isolation we note that none of them formally qualifies as a simple substitution of `find-certain-relative-of-x`. When impasses of this sort occur, BOZ attempts to match the LOP to complex

equivalence class schemas constructed from the set of simple class schemas using a composition of operators rule [4]. *Operator composition* is defined as follows:

If *f* and *g* are visual operator schemas:

```
(NLAMBDA f (a1, a2, ... an)
  (<meta-command> (<predicate> a1 a2 ... an)))
```

and

```
(NLAMBDA g (b1, b2, ..., bm)
  (<meta-command> (<predicate> b1 b2 ... bm)))
```

then the *composition* of f and g, f • g, is the schema:

```
(NLAMBDA f-composition-g (a1, ... (<predicate> b1 b2 ... bm), ... am)
  (<meta-command> (<predicate> a1 (<predicate> b1 b2 ... bm) ... an))
```

for some argument, a_i.

Hence, a *composition* of two operators is a single operator that uses another operator as one or more of its arguments.

By adding the rules for defining complex schemas, BOZ can construct more complex visual operators to match complex logical operators such as the "relatives" examples above. The following shows how two visual operators, `read-label` and `find-connectee`, are composed to produce a match for `find-certain-relative-of-X`.

```
(NLAMBDA read-label (<object>)
  (ASK (Label (find-connectee <object> <OBJECT>) <LABEL>)))
```

Figure 7 shows the classifications for the logical operators in the airline reservation task. Each operator in the task can be matched by a single equivalence class. For each logical operator in the task, a set of visual operators initially qualify as substititions. For example, the list of visual operators associated with the **search** equivalence class are proposed as substitutions for the `findFlightWithOrigin` and `findSeat` operators. Similarly, the operators of the **subtraction** class match the description of the `computeLayover` operator. It is important to note that BOZ has not yet decided *which* visual operator to

choose in each case. Decisions about which visual operators to match with each logical operator are subject to further constraints computed by the visual data structuring and visual operator selection components described in Sections 5 and 6. Consequently, what BOZ has produced at this stage is a space of visual procedures that may be selected according to additional design criteria.

```
findFlightWithOrigin (search)
findDestination (lookup)
available? (lookup)
determineDeparture (lookup)
determineArrival (lookup)
computeLayover (subtraction)
layoverLessThanX? (lessthan)
determineCost (lookup)
addCosts (addition)
costLessThanX? (lessthan)
findSeat (search)
emptySeat (lookup)
```

Figure 7: Operator Classifications for the Airline Reservation Task

5. VISUAL DATA STRUCTURING

The visual data structuring component is a graphic design technique used to implement the second type of cognitive advantage of graphical displays: that graphical displays sometimes allow users to spend less time searching for needed information. The visual data structuring component examines the information required to perform each logical operator in a task. Two types of analyses are performed using this information. First, by noting the domain set that each LOP predicate is defined over, it is determined what information should appear in a graphic designed to support a task. Second, by analyzing the relationships between the operators in a task in terms of the domain sets of information they manipulate, it is determined: (a) how information shared by several operators should be collected in the same spatial locality and visually encoded using the same primitive graphical languages; (b) and how information not shared among operators can be partitioned into distinct displays. The visual data structuring component produces a *visual data structure specification* that outlines the displays that will be used to support the task, the information should appear in each display, and how information is to be grouped together within each display. The visual data structuring component does *not* decide how information is to be graphically encoded in the display. These decisions are made by the visual operator selection component (Section 6).

The remainder of the section describes a scheme that analyzes relationships between operators by representing each operator as a vector defined over a set of domain sets. Relationships between vectors are determined by identifying common domain sets occuring in vectors. A complete sketch of all relationships between vectors reveals how information is to be collected together into graphical objects and partitioned among displays.

5.1 Operator Vectors

Recall that every task description is defined over a finite collection of domain sets. When taken together, all of the domain sets used by a task description form a feature space. A *feature space* is formally defined as the cross product of all domain sets spanned by a task description. Figure 8 shows an example of a feature space defined over the domain sets that pertain to the airline reservation task.

[flight] x [origin] x [destination] x [departure] x [arrival] x [layover] x [seat]

Figure 8: Feature Space for the Airline Reservation Task

Each individual operator in a task description computes a relation of the form (p, o, a_i, ... a_j) or (f, a_i, ... a_j) for some i and j less than or equal to the total number of domain sets drawn from a feature space, called a *vector*. Vectors of the first form are called *search vectors*. Vectors of the second form are called *computation vectors*. The first element in a search vector, p, names the predicate that the operator computes. The second element in a search vector, o, is known as the *object* of the vector. The first element in a computation vector names the *function* that the operator computes. The remaining elements in either kind of vector, a_i, ... a_j, are refered to as *attributes*.

The operators in the airline reservation task define the following vectors:

1. findFlightWithOrigin = ({flight} x {origin})
2. findDestination = ({flight} x {destination})
3. available? = ({flight} x {availability})
4. determineDeparture = ({flight} x {departure})
5. determineArrival = ({flight} x {arrival})
6. determineCost = ({flight} x {cost})
7. findSeatOnFlight = ({flight} x {seat})
8. emptySeat? = ({seat} x {availability})
9. computeLayover = ({departure} x {arrival} x {layover})

10. layoverLessThanX? = ({layover} x {layover})

11. addCosts = ({cost})

12. costLessThanX? = ({cost})

5.2 Relationships Between Vectors

Relationships between vectors are determined in the following way. Let OP be a set of operators occuring in a task description, and op_i an arbitrary member of OP. Let $V(op_i)$ be the vector implied by an operator op_i. The function $R(V(op_i))$ computes four types of relationships between the vectors $V(op_i)$ for all i. Search vectors sv_1 and sv_2 are said to be *conjoint* when they contain common objects, o. Search vectors $sv_1 = (p, o, a_1, ... a_i)$ and $sv_2 = (p, o, b_1, ... b_j)$ are *parallel* when there exists a computation vector $cv = (f, c_1, ... c_k)$ such that there exists some c_m in $a_1, ... a_i$ and some c_n in $b_1, ... b_j$ for m and n in (1 ... k). Search vectors sv_1 and sv_2 are said to be *orthogonal* if some attribute, a_i, of sv_2 appears as the object, o, in sv_1. Search vectors sv_1 and sv_2 are *disjoint* when they are neither parallel or orthogonal.

Relationships between vectors are used to define visual data structures in the following way. *Conjoint vectors* group together attributes that pertain to the same object. Consequently, these attributes should be encoded in a single graphical object in order to reduce eye movement over the display when searching for that object and its attributes. *Parallel vectors* indicate that some visual operator(s) requires that two or more different objects and their attributes be coordinated in order to draw a particular inference. Consequently, both objects should appear in the same display and be encoded using the same primitive graphical language. *Disjoint vectors* indicate that no visual operator requires that two or more objects be coordinated to drawn an inference. Vectors of disjoint operators can be supported in different displays since the information they manipulate is never used together. *Orthogonal vectors* indicate part-of relationships between objects. Information manipulated by orthogonal vectors is presented in separate nested displays. That is, the user should be able to view the part-of display by making an appropriate selection in the first display.

Figure 9 shows the relationships between the vectors in the airline reservation task.

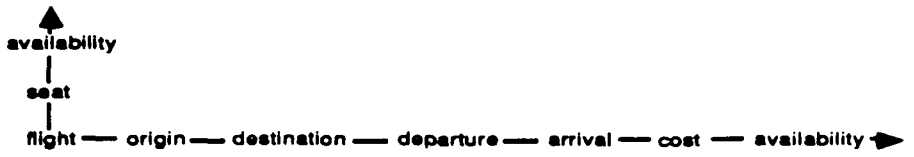


Figure 9: Vector Relationships for the Airline Reservation Task

Since vectors 1 through 7 are conjoint, information pertaining to the origin, destination, departure, arrival, cost, and availability of a flight form a single vector and should be encoded in the same graphical object. Vectors 7 and 8 are also conjoint, hence, all information about seats should be similarly encoded in the same graphical object. The two conjoint vectors are orthogonal indicating that each seat object is a part of a flight object. Hence, seating information should be presented in a separate display that is nested inside each flight box.

The initial visual data structure specification for the airline task is shown in Figure #.

```
(NESTED (DISPLAY1 (flight (Origin Destination Departure
                           Arrival Cost Availability))
        (DISPLAY2 (seat (Seat Availability))))
```

Figure #: Initial Visual Data Structure Specification for the Airline Reservation Task

The predicates that have been grouped with each object are precisely those that will be encoded in the graphic. Note that information about flight numbers and layovers will not be encoded in any graphic. This has occured since information about flight numbers is never used in the task, and facts about layovers are produced as the results of a visual operator, `compute-layover`. It is especially important to note that it is not yet been decided how facts about the origin, destination, departure, etc. of a flight are to be visually encoded in the graphic. That is, BOZ has not yet associated the names of primitive graphical languages with the predicate names appearing in the visual data structure specification. Which primitive graphical languages to associate with each predicate is determined by the visual operators selected to substitute the logical operators in the task.

6. VISUAL OPERATOR SELECTION

The visual operator selection component chooses a single visual operator to substitute each logical operator from the list of possibilities generated by the visual operator substitution component. Selecting a single visual operator to substitute each logical operator accomplishes two things: (a) reduces the space of possible visual procedures to a single

visual procedure judged to be the most effective; and (b) allows BOZ to design a single accompanying graphic that supports human performance of the selected visual procedure.

Three important issues constrain the selection of visual operators. First, since the goal is to arrive at a visual procedure that minimizes the effort required to correctly complete a task, for each logical operator we wish to choose that visual operator that is performed most efficiently and accurately by human users. A first criteria for operator selection involves estimating the relative performance efficiency and accuracy of the visual operators. Second, recall that each visual operator is associated with a primitive graphical language that must be used to graphically encode information manipulated by that operator. A second criteria when selecting operators is that the representational power of the primitive graphical language associated with a candidate visual operator is sufficient to encode the relational facts manipulated by the operator. Third, recall that the visual data structure specification produced by the visual data structuring component constrains some domain sets of information to be represented in a single graphical object, or using the same primitive graphical language. A third criteria for operator selection is that the primitive graphical languages associated with the selected visual operators be combinable such that they result in coherent graphical representations that agree with the visual data structure specification for the task.

6.1 Human Performance Rankings for Visual Operators

The most important criteria when selecting a visual operator is choosing that operator that allows the human user to obtain the results of the operator most efficiently and accurately. To determine which of a set of visual operators is likely to be the most performance effective, BOZ uses a two-tier ranking system that is a generalization of the approach used in Mackinlay's APT program [25]. The first tier ranks the equivalence classes for operators appearing in Table III in order of their relative difficulty. For instance, **search** operators require more effort to perform than **lookup** operators. Consequently, they are always awarded the most efficient visual operators. The second tier ranks the visual operators within each operator class. For instance, determining the horizontal distance between two points on a scale is generally performed more efficiently than determining the difference between two sloped lines. The rankings were generated using a combination of two methods: (a) theoretical predictions based on a more fine-grained consideration of each visual operator [5, 33]; and (b) experimental observations of human perceptual task performance [11, 19, 31, 34]. Table V shows the rankings for visual operator equivalence classes, and the rankings for the visual operators in each class.

TABLE V
Ranking of Visual Operators and Equivalence Classes

A. Class Rankings:

1. plus, difference, quotient, times
2. verify
3. search
4. max, min
5. lessthan, greaterthan
6. equal
7. search and lookup
8. lookup

B. Operator Rankings:

SEARCH OPERATORS

search, verify: {Visibility, HorzPos, VertPos, Shape, Connectivity, Shading, Height, Width, Slope, LineDashing, LineLength, LineThickness, Labels, Area}
lookup, search and lookup: {Shading, Shape, Labels, Height, Width, Slope, LineDashing, LineThickness, Connectivity, HorzPos, VertPos, LineLength, Area, Visibility}

COMPUTATION OPERATORS

(set)equal: {Labels, Shading, HorzPos, VertPos, Shape, LineDashing, Height, Width, LineThickness, LineLength, Slope, Connectivity, Visibility, Area}
(set)lessthan, (set)greaterthan: {Shading, HorzPos, VertPos, Height, Width, Slope, LineThickness, LineLength, Labels, Connectivity, Shape, LineDashing, Visibility, Area}
(set)plus, (set)times: {Height, Width, LineLength, LineThickness, HorzPos, VertPos, Labels, Slope, Connectivity, Shading, LineDashing, Shape, Area}
(set)difference, (set)quotient: {HorzPos, VertPos, Height, Width, LineLength, LineThickness, Labels, Connectivity, Slope, Area, Shading, Shape}
max, min: {Visibility, HorzPos, VertPos, Connectivity, Shading, Height, Width, Shape, LineDashing, Slope, LineLength, LineThickness, Labels, Area}

6.2 Expressiveness

The second criterion used during operator selection is that a selected visual operator must be associated with a primitive graphical language that is powerful enough to encode the relational facts manipulated by that operator. For example, even though the `search-shaded-object` is the most efficiently performed search operator, it cannot be selected to substitute the `findFlightWithOrigin` logical operator since the number of different cities exceeds the number of different shades. When a selected visual operator fails to meet the expressiveness needs of a logical operator it is disqualified and the next highest ranking visual operator is considered. The interested reader can consult Mackinlay [25] for a thorough analysis of primitive graphical language expressiveness. BOZ, like all other recent presentation systems, adopts Mackinlay's mechanism for deciding expressiveness.

6.3 Operator Combinability

The third criterion for operator selection concerns the combinability of visual operators. The following example illustrates the notion of combinability of visual operators. Suppose we have selected the `stack-heights` visual operator to substitute the `addCosts` logical operator in the airline reservation task and are currently selecting a visual operator to substitute the `findFlightWithOrigin` operator. Suppose that we are currently

considering the `determine-slope` visual operator as a candidate selection. Recall that the visual data structuring component has indicated that the information relevant to these two operators should be encoded in the same graphical object. Every visual operator has associated with it a *graphical presentation object* that is used to graphically encode the information manipulated by that object. For example, the graphical presentation object for the `stack-heights` and `determine-slope` operators are <rectangle> and <line>, respectively. Note that the information relevant to the two operators cannot be encoded in the same graphical object. That is, it is meaningless to speak of the slope of a rectangle or the height of a line. Consequently, these two operators are not combinable and we must disqualify `determine-slope` as a candiate for selection. Now suppose we move on and consider the `read-label` visual operator. Note that the two operators are indeed combinable. Even though the graphical presentation object for the `read-label` operator is <string> and the graphical presentation object for the `stack-heights` operator is <rectangle>, the two graphical objects can be combined to form a labeled rectangle.

The next two sections describe how the set of graphical presentation objects and a set of graphical object composition rules are used by BOZ to decide combinability of visual operators.

6.3.1 Graphical Presentation Objects

Each primitive graphical language has a *graphical presentation object* associated with it, either: <point>, <line>, <rectangle>, <polygon>, or <string>. The graphical presentation objects for a primitive graphical language are those graphical objects that support the performance of the visual operators that are associated with that graphical language. For example, the graphical presentation object for the Height primitive graphical language is <rectangle>. Note that only this object makes the visual operators associated with the Height language meaningful. That is, it would be impossible to determine the height of a point since a point by definition has no spatial extent.

Table VI lists the graphical presentation objects associated with each of the primitive graphical languages.

TABLE VI
Graphical Presentation Objects for the Primitive Graphical Languages

Horizontal Position = <point>
Vertical Position = <point>
Cartesian Position = <point>
Height = <rectangle>
Width = <rectangle>
Line Length = <line>
Area = <polygon>
Connectivity = <line>, <point>
Shading = <polygon>, <rectangle>
Labels = <string>
Color = <point>, <line>, <rectangle>, <polygon>
Line Thickness = <line>, <rectangle>, <polygon>
Line Dashing = <line>, <rectangle>, <polygon>
Slope = <line>
Shape = <polygon>
Visibility = <point>, <line>, <rectangle>, <polygon>, <string>

The first step in deciding visual operator combinability is to determine the graphical presentation object of a candidate visual operator.

6.3.2 Composition Rules for Graphical Presentation Objects

The second step in deciding operator combinability is to compare the graphical presentation object of the visual operator currently being considered with the presentation objects of all previously selected operators that appear in the same vector in the visual data structure specification. If the graphical presentation object matches those of the previously chosen operators then the new operator is combinable. If the presentation object does not match, BOZ attempts to show them combinable using the set of composition rules for graphical objects given in Table VII.

TABLE VII
Composition Rules for Graphical Presentation Objects

Mark Composition Rules:
RULE 1: <point> + <point> = <point>
RULE 2: <point> + <line> = <line>
RULE 3: <line> + <line> = <line>
RULE 4: <rectangle> + <point> = <rectangle>
RULE 5: <rectangle> + <rectangle> = <rectangle>
RULE 6: <polygon> + <point> = <polygon>
RULE 7: <polygon> + <polygon> = <polygon>
RULE 8: <label> + <label> = <label>
RULE 9: <label> + <line> = <line>
RULE 10: <label> + <rectangle> = <rectangle>
RULE 11: <label> + <polygon> = <polygon>

Axis Composition Rules:
RULE 12: <horz-axis> + <horz-axis> = <horz-axis>
RULE 13: <vert-axis> + <vert-axis> = <vert-axis>
RULE 14: <horz-axis> + <vert-axis> = <cart-axis>

Network Composition Rules:
RULE 15: <node-link-node> + <node-link-node> = <node-link-node>

Each composition rule describes how a set of individual presentation objects can be legally composed to form a single presentation object that inherits all of the graphical properties of the constituent objects. For any new visual operator and set of previously selected operators, the new operator is combinable if and only if a rule can be found that maps the set of presentation objects into another legal presentation object.

Applying the operator selection strategy to the set of possible visual operators obtained for the airline reservation procedure yields the visual procedure shown in Figure 11. Note that every logical operator has been replaced with a single visual operator.

```
(TASK airlineReservation

(PROCEDURE
 (repeat
  (if (search-object-with-label FLIGHT 'pit) then
   (if (shaded? flight) then
    (read-label flight DESTINATION)
    (determine-horz-pos flight ARRIVAL)
    (repeat
     (if (search-object-with-label CONNECTING destination) then
      (if (shaded? connecting) then
       (determine-horz-pos connecting DEPARTURE)
       (determine-horz-distance departure arrival LAYOVER)
       (if (smaller-horz-interval? layover) then
        (determine-height flight COST1)
        (determine-height connecting COST2)
        (stack-heights cost1 cost2 TOTAL)
        (if (shorter? total) then
         (repeat
          (search-object-with-label flight SEAT1)
         (until (NOT (shaded? seat1))))
         (repeat
          (search-object-with-label connecting SEAT2)
         (until (NOT (shaded? seat2))))))
```

```
      (until done))))
(until done))
```

Figure 11: The Final Visual Airline Reservation Procedure

Figure 12 shows the final visual data structure specification for the airline reservation task after the composition rules have been applied.

```
(NESTED (DISPLAY1 (flight ((Origin Labels) (Destination Labels)
                           (Departure HorzPos) (Arrival HorzPos)
                           (Cost Height) (Availability Shading))
                           <rectangle>)
        (DISPLAY2 (seat ((Availability Shading)) <rectangle>)))
```

Figure 12: The Final Visual Data Structure Specification for the Airline Reservation Task

Note that each predicate appearing in the visual data structure specification has been associated with a single primitive graphical langauge. In each case the primitive graphical langauge chosen is precisely that language associated with the visual operators that have been selected to manipulate that type of information.

6.4 Limitations of Automated Visual Operator Selection

It is important to note that there exists no algorithmic strategy that always chooses the most efficiently or accurately performed visual operator, including those based on experimental observations and detailed theoretical predictions. I am aware of no experimental result of people using graphics that has been successfully generalized across any one of: user [20], level of skill [16], practice [29], task [18], particular display used [24], age [9], culture [17], or even social situation [1]! Each of these factors have been shown to introduce variance strong enough to overturn the results of any particular experiment, making strong generalizations of these results inappropriate. What we can hope to achieve in an automated graphic design tool is a codified set of operational design principles that perform satisfactorily across interesting tasks and graphics.

Many task domains make use of domain-specific graphic conventions for which practitioners of that domain have acquired practiced skill in using. It is not always the case that these conventions were chosen based on which conventions were the most cognitively efficient, but rather what informally seemed to comprise a useful notation at the time the graphic convention was designed. Without specific knowledge of a problem domain, an automated graphic design tool is unable to identify and select operators that correspond to existing graphic conventions and this is a second limitation of automated graphic design.

BOZ, like APT [25], allows the designer to intervene and manually select visual operators in order to support existing conventions.

The human performance efficiency of a visual operator may be sensitive to the particular data on which the operator is performed. For example, judging the distance between two points on scale that are aligned three units away from one another appears to be easier than if the points are aligned, say, thirty-nine units apart. This phenomenon occurs because some input data allow the user to exploit more low-level perceptual capabilities such as *subitizing* [21]. A more reliable visual operator ranking system might be achieved by making the set of rankings functionally dependent on the set of relational facts to be displayed. That is, rather than using a fixed set of operator rankings, a different set of rankings would be computed for each different set of relational facts. The usefulness of such a scheme, along with its computational feasability, is an open question.

BOZ's visual operator selection component proceeds under the assumption that the time to perform a visual operator is unaffected by any other visual operators that are performed before or after that operator. That is, the time to perform a visual operator is assumed to be context-independent. There are two ways in which this assumption may be invalidated. First, while the question of parallel perceptual task performance remains open for debate, experimental observations suggest that human user are sometimes able to obtain the results of several visual operators in a time less than the sum of the observed performance times for each individual operator [33]. Furthermore, researchers have gained a partial understanding of which combinations of visual operators exhibit this property. Combinations of visual operators that achieve this property result in a greater savings in performance efficiency. Casner and Larkin [8] explore the topic of operator parallelization in more detail. Second, experimental observations of users in other task domains such as typing suggest that certain combinations of operators may sometimes result in a performance time that is greater that the sum of the individual performance times [35].

7. VISUAL DISPLAY RENDERING

The rendering component uses the visual data structure specification to translate relational facts submitted with the logical procedure to visual facts. *Visual facts* are graphical encodings of the original set of relational facts, presented using a graphic design that agrees precisely with the visual data structure specification for the task. The rendering component produces a fully rendered graphical display of the visual facts. Graphical displays produced by the rendering component support interactive manipulations of the graphical

objects appearing in the display, allowing users to effect changes in the internally stored relational facts by making changes to the visually displayed facts.

7.1 Translating Logical Facts to Structured Visual Facts

A prerequisite to graphically rendering arbitrary sets of relational facts on the computer screen is a notation for representing visual facts that is equivalent to the notation used to express relational facts. To accomplish this, Mackinlay's formalism for expressing visual facts is used, and this has been shown to be informationally equivalent to a logical representation of the same relational information [25]. Mackinlay's formulation allows relational facts to be expressed using each of the primitive graphical languages given in Table I. Visual facts expressed in a primitive graphical language take the following form: `(PGL <OBJECT> <VALUE>)`. For example, the facts in Figure 13 describe a square-shaped graphical object that is shaded black and positioned along a horizontal axis.

```
(object obj001)
(shape obj001 square)
(horzpos obj001 6)
(shading obj001 black)
```

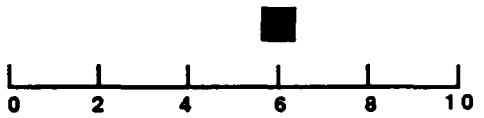


Figure 13: Example Visual Facts

Figure 14 shows how relational facts (left side) about airline flights are renamed to graphical facts (right side) when the mapping given in the visual data structure specification is applied.

```
(origin flight117 pit)                    (label flight117 pit)
(origin flight239 hou)                    (label flight239 hou)
(destination flight117 hou)               (label flight117 hou)
(destination flight239 mex)               (label flight239 mex)
(departure flight117 10:00)              (horzpos flight117 10)
(departure flight239 15:00)              (horzpos flight239 15)
(arrival flight117 12:50)              (horzpos flight117 12.83)
(arrival flight239 17:15)              (horzpos flight239 17.25)
(cost flight117 179)                     (height flight117 1.79)
(cost flight239 239)                     (height flight239 2.39)
(availability flight117 ok)          (shading flight117 whiteshade)
(availability flight239 ok)          (shading flight239 whiteshade)
```

Figure 14: Translated Airline Reservation Facts

A final notation is needed for expressing collections of visual facts whose structure agrees with the visual data structure specification for the task. A *structured fact* corresponds to the

"gestalt wholes" defined by the visual data structure specification. For example, the following structured facts show how the visual facts in Figure 14 are structured according to the visual data structure specification for the airline task given in Figure 12.

```
(((labels flight117 pittsburgh)
  (labels flight117 hou)
  (horzpos flight117 10)
  (horzpos flight117 12.83)
  (height flight117 1.79)
  (shading flight117 whiteshade))
((labels flight239 hou)
  (labels flight239 mex)
  (horzpos flight239 15.00)
  (horzpos flight239 17.25)
  (height flight239 2.39)
  (shading flight239 whiteshade)))
```

7.2 Rendering Visual Facts

The rendering component automatically displays arbitrary sets of structured visual facts on the computer screen. This is accomplished by considered each structured fact, determining the form in which it is to be presented by consulting the visual data structure specification, and rendering the image of the fact on the screen. The rendering algorithm uses an object-oriented approach to rendering graphical presentation objects and their graphical properties. For every type of graphical presentation object (i.e., <point>, <line>, <rectangle>, <polygon>, and <string>) there exists a corresponding *display object* that can be rendered on the screen. To expedite the rendering of display objects, drawing primitives are used to create images rather than bitmap displays. Display objects can inherit one or more of a set of *display methods* that render the graphical properties of a display object. Display methods are defined for each of the primitive graphical languages in Table I. For displays that do not use horizontal and vertical position to encode information, a simple displacement scheme is used to avoid occlusion of display objects by other display objects. Scales and guidelines are automatically computed, drawn, and labeled using the DOMAINSETS field in the logical procedure description. Fonts have been chosen arbitrarily and standardized. Nested graphics are implemented by mouse-sensitive buttons that are always placed in the lower left corner in rectangles and polygons, and immediately on top of points and lines. Customized methods are automatically attached to the buttons that cause the nested graphic to be rendered when the button is selected.

Figure 15 shows a fully rendered set of visual airline facts. As specified by the visual data structure specification, the display consists of a single type of graphical object (i.e., a flight

box) that inherits four graphical properties (i.e., horizontal position, shading, height, and labels).

****
Figure 15 here
****

Selecting the seats button for any flight causes the nested seating chart for that flight to be rendered. A rendered seating chart is shown in Figure 16.

****
Figure 16 here
****

The graphics generated by the rendering component support two-way interactions between sets of relational facts and their visual images. In addition to being able to effect changes in a graphical display through manipulations of the internally-stored relational facts, the graphical objects in the displays can be manipulated by the user to effect changes in the set of stored relational facts. For example, the upper, left-most flight box in the display in Figure 15 indicates that there is flight from PIT to JFK leaving at 11:30am with no available seats costing $400. The user may simultaneously change the graphical and internal representation of this information by simply mouse-selecting the flight box and moving it to a new location, changing its shading, or increasing or decreasing its height. Casner [6] generalizes this technique in a tool that allows users to create customized diagramming languages that can be attached to and used to manipulate internally stored data and knowledge representation structures.

7.3 Limitations of Automated Rendering

There are several important limitations of the automated rendering component. First, the rendering component is incapable of rendering displays that make use of domain-specific conventions. For example, airline seating charts typically orient the aircraft pointing east, lower numbered seats appearing the right and higher number seats to the left. Since BOZ's rendering component has no knowledge of this convention, the seats are arranged in increasing order from left to right as are the hours along the time scale. Second, many displays depict realistic information such as spatial arrangements and shapes that do not encode information vital to the task at hand but preserve many features of a real-world artifact in an artificial representation. For example, airline seating charts typically depict the aisle separating the two halves of the plane. Some seating charts also use chair-shaped

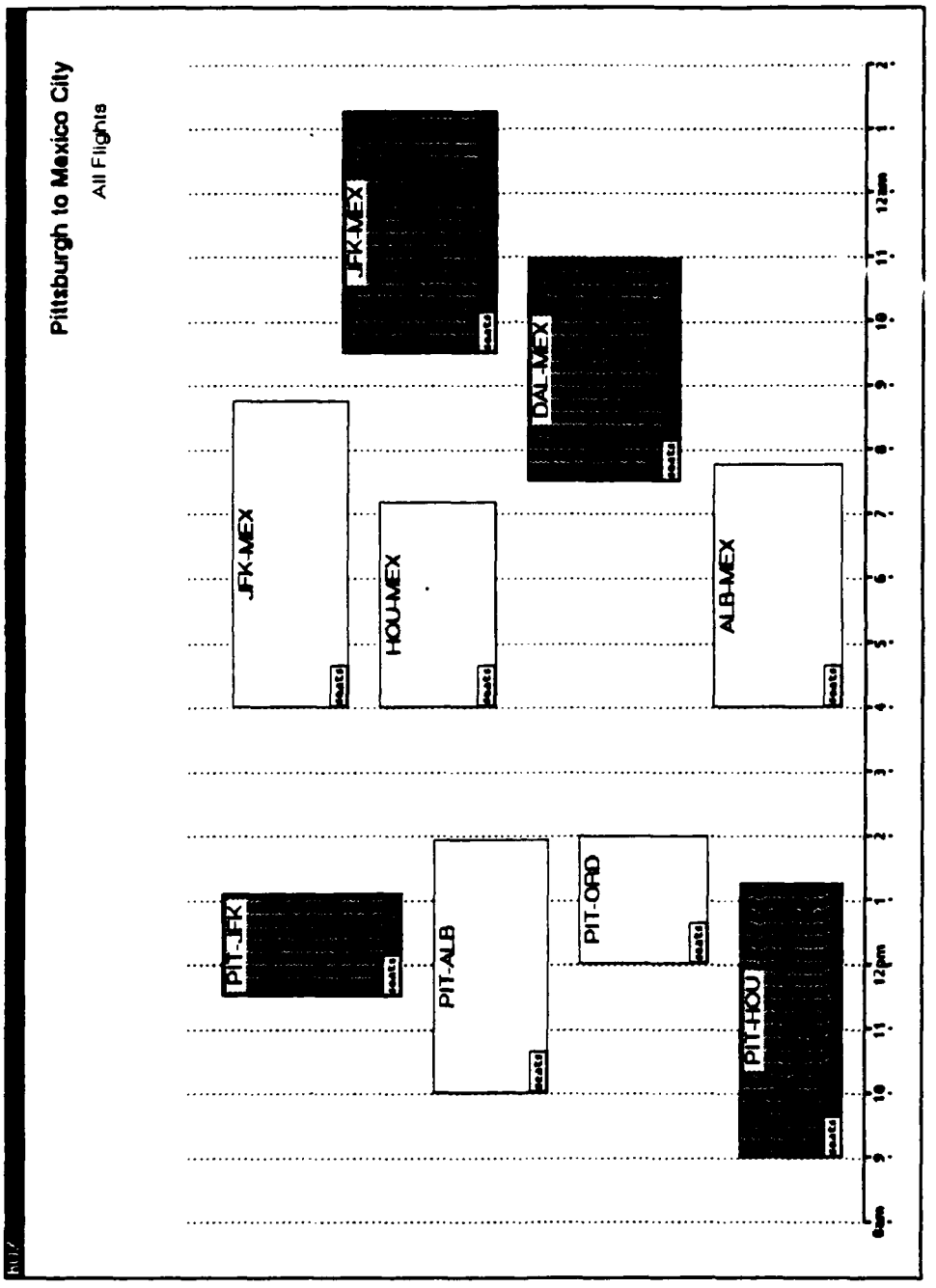


Figure 15: Rendered Airline Graphic

Window Image

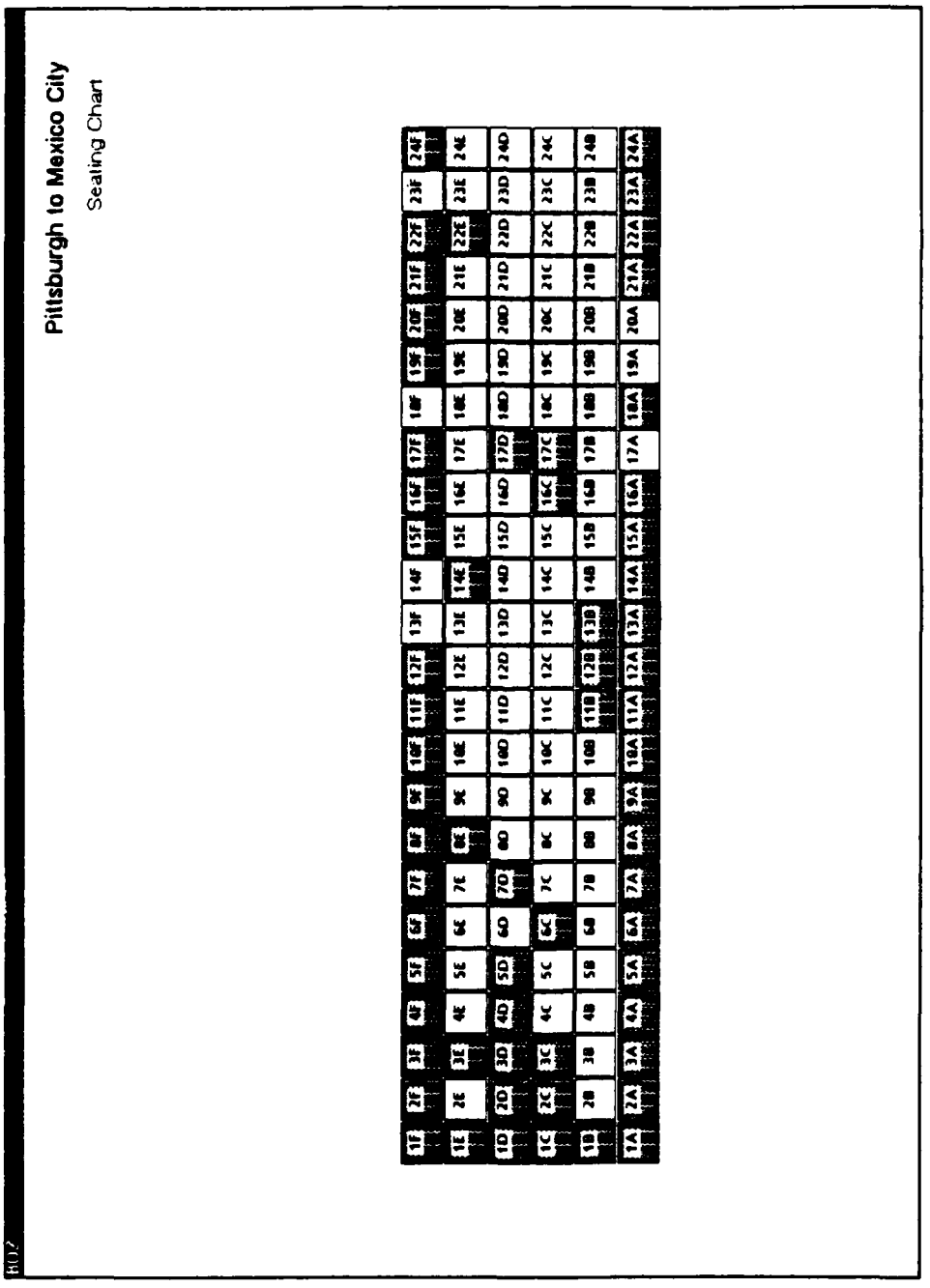


Figure 16: Rendered Seating Chart

icons to represent seats instead of the generic box-shape used in BOZ's display. BOZ of course has no knowledge of these conventions. Note that despite these two limitations it is still possible to locate any seat. What may be lost is a familiarity and practice that users may have already acquired using other conventions.

8. COGNITIVE ANALYSIS OF THE AIRLINE GRAPHICS

Table VIII summarizes the potential cognitive efficiencies of the airline reservation graphics. The efficiencies occur in two forms that agree with Larkin and Simon's theoretical analysis and the design goals of BOZ: (a) ways of substituting efficient visual operators in place of more demanding logical operators; and (b) ways of reducing eye movement and the number of items considered when searching for needed information.

TABLE VIII
Predicted Cognitive Efficiencies of the Airline Graphic

Operator Substitutions
Substitutes a distance judgement (`determine-horz-pos`) in place of subtracting numerically expressed departure and arrival times (`compute-layover`).
Substitutes a shade judgement (`shaded?`) for reading the words "ok" and "full" (`available?`).
Substitutes judging the combined heights of two flight boxes (`stack-heights`) for adding two numerically expressed costs (`addCosts`).
Search Advantages
Eliminates eye movements when looking up time, city, cost, and availability information since this information is represented in the same spatial locality (a single flight box).
Allows users to limit their search for connecting flights to only those flights that appear to the right of the originating flight.
Since shading can be processed pre-attentively, users may immediately exclude from their search any flight square that has no available seats.
Allows users to immediately rule out "tall" flights from their search since these are likely to exceed the $500 limit.
When looking for an available seat, users can immediately eliminate shaded seats. Users can also restrict their search to window or aisle seats by following simple eye movement patterns.

It is important to note that the hypothesized advantages of any BOZ-designed graphic depend on the user understanding and being able to perform the visual procedure supported by that graphic. Whether or not real users can or actually do follow the designed visual procedure, and the extent to which the predicted efficiency advantages are reflected in users' performance is an empirical question to which we now turn our attention.

9. USERS' PERFORMANCE WITH THE AIRLINE GRAPHIC

Casner and Larkin [7] describes an experimental study designed to determine the extent to which the hypothesized cognitive efficiency advantages of the airline graphic shown in Table VIII were actually reflected in users' performance. To better understand the contribution made by each hypothesized efficiency advantage, a sequence of four graphics was designed in which each successive graphic provided an additional opportunity to substitute a visual for logical operator, and an additional opportunity to reduce visual search. The final display in the sequence contained all of the efficiency advantages listed in Table VIII (excepting those pertaining to the seating chart display). The four experimental graphics, herein called Graphics 1, 2, 3, and 4, are shown in Figure 17.

****
Figure 17 here
****

Response times were collected from eight participants who performed the airline reservation task using the four graphics a total of ten times each (forty trials total). Users completed the task after receiving an explanation of the conventions used in each graphic and one practice trial. The order in which the graphics were presented to the users was varied systematically to evenly distribute effects due to learning and practice.

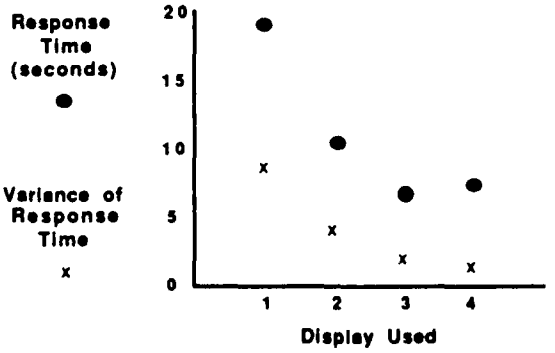
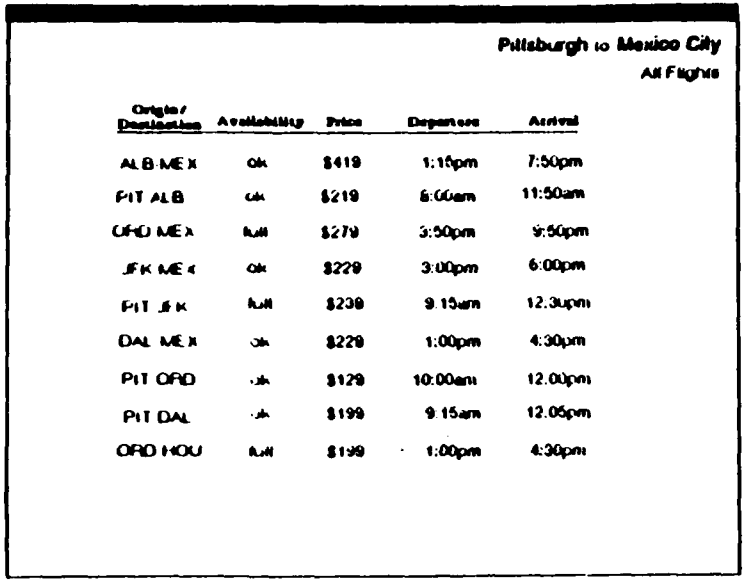
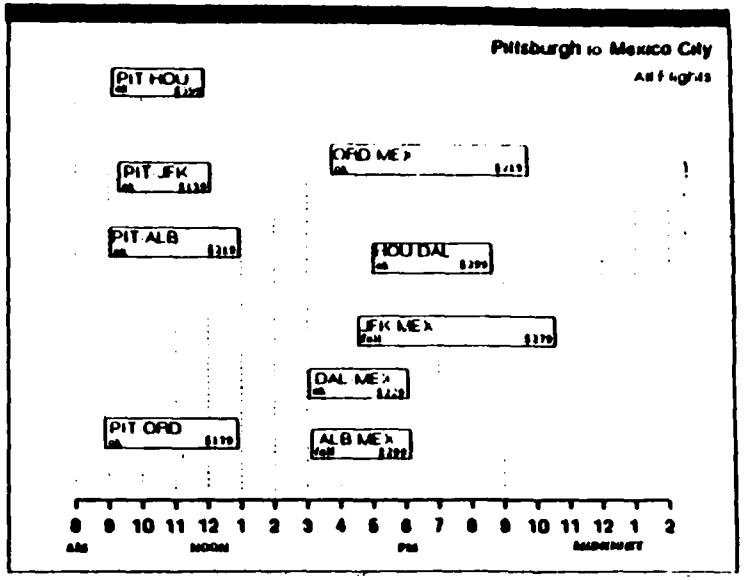
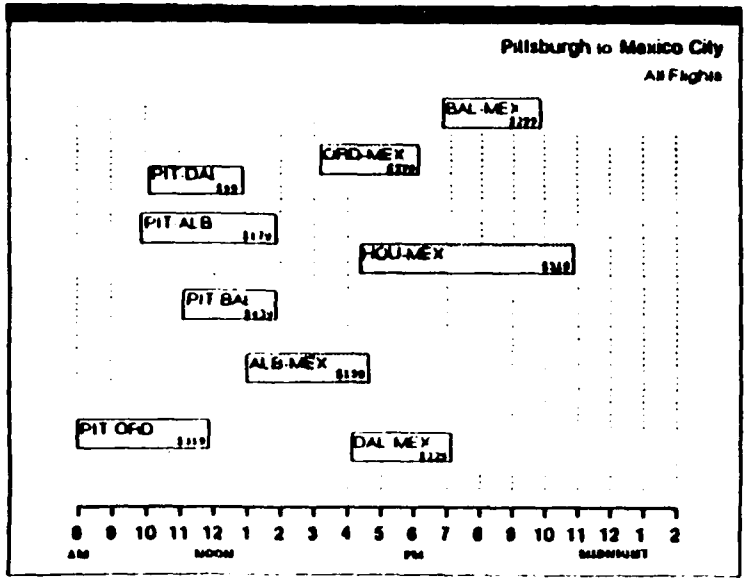


Figure 18: User's Performance Times for the Airline Reservation Task

The results shown in Figure 18 indicate significant differences in response times between Graphics 1 and 2, and between Graphics 2 and 3, but not between Graphics 3 and 4. The data suggest that time scale encoding used in Graphic 2 (and also in Graphics 3 and 4) reduced the amount of time required to locate two connecting flights, and to determine whether or not two flights obey the layover constraint. Allowing users to perform the







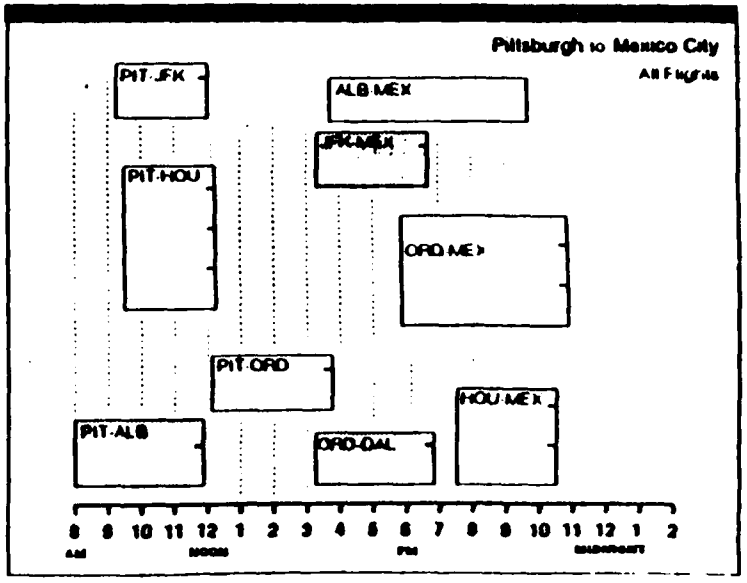


Figure 17: Four Experimental Graphics

perceptual operator of determining the shade of a flight box (Graphic 2) also resulted in a significant savings. The perceptual task of determining whether or not two flights obey the cost constraint by judging the heights of the flight squares did not result in any reliable savings over the task of adding the two numbers, or in narrowing down the search space of flights to consider. An analysis of the standard deviations in response times suggests that users exhibited significantly more stable performance between Graphics 1, 2, and 3 in that order.

Our next step was to understand how users obtained the efficiency savings we observed. We ran a regression analysis on the number of times each operator must be performed using each graphic, the number of flight boxes searched, and the observed user response times. We obtained the best fitting models when each graphic was combined with the procedures that used all of the operator substitutions and search reductions that were applicable to that graphic. This suggests that for each graphic, users took advantage of all of the operator substitutions and search reductions that were possible with that graphic. The beta-coefficients in the regression model yielded estimates on performance time for several of the individual visual and logical operators.

- The time required to fix the eye on each item in a graphic was uniformly about 330 milliseconds for all four graphics.

- Visually estimating layovers (`determine-horz-distance`) using Graphics 2, 3, and 4 proceeded about 2 seconds faster than subtracting the numerically expressed times.

- Judging the combined heights of two flight boxes (`stack-heights`) in Graphic 4 was negligibly 100 to 300 milliseconds slower than adding the numerically expressed costs.

The savings gained through substitution of visual for logical operators and use of search reductions match well with the global reductions observed in overall response times. Overall the results agree with users' comments after using all four graphics: that Graphics 3 and 4 were the most effective. The interested reader can find details of the experimental design and methodology in Casner and Larkin [7].

10. GENERAL DISCUSSION

The research described above proposes a task-analytic approach to the design of information graphics in which graphics are viewed as perceptually and graphically manipulated data structures that help streamline task performance much in the way that abstract data structures help expedite abstract computational processes. The important distinction made in task-based graphic design is that the effective use of visual data structures, as with abstract data structures, depends on designing the right structure for a given task. That is, it is inappropriate to say that a particular graphic is the best choice or that it is useful in general. Consequently, the design methodology embodied in BOZ proceeds by analyzing the operators that comprise a user task and generating informationally equivalent visual tasks that can be performed more efficiently by humans. The design of accompanying visual displays of information is targeted primarily at supporting efficient and accurate human performance of the visual task. The examples and experimental results show how the task-analytic approach can be successfully applied to designing effective visual tasks and displays that provide two types of cognitive efficiency advantages: (1) allowing users to substitute perceptual inferences in place of more demanding logical inferences; and (2) reducing user's search for needed information.

Real-Time Automated Graphical Presentation. Aside from an algorithmic specification of a design theory for information graphics, BOZ appears potentially useful as a tool for the automated design and generation of graphical displays in computer information systems. However, two limitations of the present model prevent BOZ's current use in real-time applications. First, the descriptions of logical procedures required by BOZ as input must presently be hand-generated. A future research topic is to investigate ways of automatically generating procedural descriptions, elimintating the need for human intervention. SAGE [27] uses a discourse processor that allows descriptions of simple operators to be generated by analyzing simple natural language queries made by the user. However, this approach is unable to generate descriptions of complex procedures defined using collections of many operators. Second, while the runtime complexity of BOZ may theoretically be able to meet the demands of on-line information systems, the present implementation fails to produce graphics in a time that would be considered acceptable by computer users. The rendering component is particularly slow for graphics containing many graphical objects. The search complexity for BOZ's visual operator substitution component is presently: $T_{\text{operator substitution}} = n * c$, where n is the number of logical operators appearing in a procedure, and c is the number of possible operator classes that each logical operator must be matched against. The Xerox implementation of BOZ required

about nine seconds to classify the airline reservation task operators as shown in Figure 7. BOZ's visual data structuring algorithm is linear in the number of logical operators, n, and domain sets, d.: $T_{\text{data structuring}} = n * d$. BOZ required about two seconds to design the initial visual data structure shown in Figure 10. The visual operator selection component runs nlog(n) in the number of logical operators: $T_{\text{operator selection}} = n * s$, where s is the number of operators that have been selected thus far. The visual operator selection component required seven seconds to select the visual procedure and data structure shown in Figures 11 and 12. The object-oriented rendering component is linear in the number of structured facts to be presented, f, and the number of primitive graphical languages, p, appearing in each structured fact: $T_{\text{rendition}} = f * p$. The rendering component required approximately twelve seconds to render the flights display in Figure 15 and approximately one minute fifteen seconds to generate the seating chart display in Figure 16.

Overall, BOZ designed both displays in about eighteen seconds, rendering the flights and seating charts displays after thirty seconds and one minute forty-five seconds, respectively. BOZ's current runtime does not fall within an acceptable standard for real-time data presentation. A future research topic is to investigate ways of making BOZ operate more efficiently. The rendering component is currently being reimplemented on a Macintosh II equipped with ROM-stored QUICKDRAW™ drawing primitives to investigate the effects of increased hardware and graphics support.

Executable Logical and Visual Procedures. BOZ contains an additional feature that allows the logical procedures created by the designer, and the visual procedures produced by BOZ to be compiled into executable functions. These executable functions manipulate databases of logical and visual facts such as those shown in Figures 6 and 14. This feature allows alternative procedures to be executed while the number of operator firings and items searched are counted for any combination of logical or visual procedure and graphic. These measures can be used to obtain detailed quantitative predictions on the effectiveness of any procedure and graphic produced by BOZ, and may avoid the need to perform time-consuming experimental studies with real users such as the one described in Section 9. Casner and Larkin [8] use the simulation tool to explore other cognitive advantages of graphical display-based task performance and problem solving, and to better understand the details of how reductions in mental computation and visual search are obtained through the use of graphical displays.

Discovering New Design Principles. BOZ may also be useful for explaining why existing graphic designs are successful, and to help discover clever design properties enjoyed by existing graphics that can be later incorporated into BOZ's design algorithm. Existing graphic designs can be analyzed by describing the tasks for which a graphic is known to be useful and showing that the design is such that it allows the users to perform computationally interesting visual procedures. Consider the graphical representation used in the calculus for vectors in the plane, shown in Figure 19.

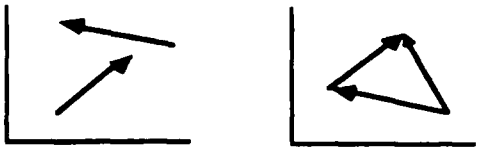


Figure #: Vectors in the Plane

Vectors use lines in the plane to represent forces acting on a body. The magnitude of a force is represented by the length of a vector line. The direction of a force is represented by the slope of a vector line. The surprising feature of vector representations is that they do not use the spatial position of a vector to encode information. The spatial position primitive graphical languages provide the most powerful and efficiently performed visual operators yet they are not used in the vector representation. The decision to keep the spatial position primitive languages "in reserve" (at some cost in cognitive efficiency) pays off when when the task of adding together vectors is introduced. Leaving the spatial position languages free of interpretation allows us to move vectors around in the plane and be sure that the vectors still represent the same information. This freedom allows us to arrange vectors such that the beginning of one vector coincides with the end of another vector as shown in the right side of Figure 19. We can now draw a line connecting the beginning of the first vector and the end of the second vector. It is easily shown that the vector added as an annotation represents the sum of the original two vectors. Summing together vectors without the benefit of the graphical representation of course requires more sophisticated mathematical knowledge and procedures.

The analysis of the design used for vectors suggests the following general design principle: sometimes a sacrifice in one aspect of a design can lead to greater gains in another aspect of the same design. The analysis of the vector representation has provided an interesting design principle that falls outside of BOZ's capabilities, and suggests a promising new idea to investigate that may be generalizable across many tasks and graphic designs.

ACKNOWLEDGEMENTS

This work is supported by the Office of Naval Research, University Research Initiative, Contract Number N00014-86-K-0678. I thank Stellan Ohlsson and Ken Koedinger for their contributions to this research.

REFERENCES

1. Asch, S.E., Studies of independence and submission to group pressure: A minority of one against a ananimous majority. *Psychological Monographs*, 1956, 70.

2. Bertin, Jacques, *Semiology of graphics*, W. Berg, transl., Madison, WI: University of Wisconsin Press, 1983.

3. Brachman, R.J. and Schmolze, J.G., An overview of the KL-ONE knowledge representation system, *Cognitive Science* 9 (2), 1985, 171-216.

4. Brainerd, W.S., and L.H. Landweber, *Theory of Computation*, New York: John Wiley and Sons, 1974.

5. Card, S.K., Moran, T.P, and Newell, A., *The Psychology of Human-Computer Interaction*, Hillsdale, NJ: Lawrence Erlbaum, 1983.

6. Casner, Stephen, Building customized diagramming languages, in *Visual Languages and Visual Programming*, S.-K. Chang, ed., New York: Plenum Press, in press.

7. Casner, Stephen, and Jill H. Larkin, Cognitive efficiency considerations for good graphic design, in *Proceedings of the Eleventh Annual Conference of the Cognitive Science Society*, Ann Arbor, Michigan, August 1989.

8. Casner, Stephen and Jill Larkin, Drawing pictures worth ten thousand words: An information-processing model of the design and use of visual displays of information, submitted to *Cognitive Science.*

9. Clancey, S. M. and W. J. Hoyer, Effects of age and skill on domain-specific visual search, in *Proceedings of the Ninth Annual Conference of the Cognitive Science Society*, Seattle, WA, 398-404.

10. Cleveland, W.S., *Elements of Graphing Data*, Monterey, CA: Wadsworth Advanced Books and Software, 1895.

11. Cleveland, W. S. and R. McGill, Graphical perception: Theory, experimentation, and application to the development of graphical methods, *Journal of the American Statistics Association* 79 (387), Sept., 1984, 531-554.

12. Fallside, David, Understanding machines in motion, PhD dissertation, Department of Psychology, Carnegie-Mellon University, May 1988.

13. Feiner, Steven, APEX: An experiment in the automatic creation of pictorial explanations, IEEE Computer Graphics and Applications, November 1985, 29-37.

14. Friedell, M., Context-sensitive, graphic presentation of information, *Computer Graphics* 16 (3), July 1982, 181-188.

15. Gnanamgari, S, Information presentation through default displays, PhD dissertation, University of Pennsylvania, May 1981.

16. Hegarty, Mary and Marcel Just, Understanding machines from text and diagram, in *Knowledge Acquisition from Text and Picture*, H. Mandl and J. Levin, eds., Amsterdam: North-Holland, 1988.

17. Hudson, W., The study of the problem of pictorial perception among accultured groups, *International Journal of Psychology* **2**, 1968, 89-107.

18. Jarvenpaa, S.L. and G.W. Dickson, Graphics and managerial decision making: Research Based Guidelines, *Communications of the ACM* **31** (6), June 1988, 764-774.

19. Jenks, C.F., and Knos, D.S., The use of shading patterns in graded series, *Annuals of the Association of American Geographers* **51**, 1961, 316-334.

20. Kieras, D., and P.G. Polson, An approach to the formal analysis of user complexity, *International Journal of Man-Machine Studies* **22**, 1985, 365-394.

21. Klahr, D., Quantification processes, in W.G. Chase (Ed.), *Visual Information Processing*, New York: Academic Press, 1973, 3-34.

22. Koedinger, K.R. and J.R. Anderson, Abstract planning and perceptual chunks: Elements of expertise in geometry, to appear in *Cognitive Science*.

23. Larkin, Jill and Herbert Simon, "Why a diagram is (sometimes) worth 10,000 words," *Cognitive Science* **11**, 1987, 65-99.

24. Lusk, E.J., and Kersnick, M., The effect of cognitive style and report format on task performance: The MIS design consequences, *Management Science* **22** (3), 1979, 787-798.

25. Mackinlay, Jock, "Automating the design of graphical presentations of relational information," *ACM Transactions on Graphics* **5** (2), April 1986, 110-141.

26. Marey, E.J., *La Methode Graphique*, Paris, 1885.

27. Roth, S.F., Mattis, J., and Mesnard, X. Graphics and natural language as components of automatic explanation. In J. Sullivan and S. Tyler (Eds.), *Architectures for Intelligent Interfaces: Elements and Prototypes*. Addison-Wesley, Reading, Mass., 1989.

28. Schmid, Calvin F., *Statistical Graphics: Design Principles and Practices*, New York: John Wiley and Sons, 1983.

29. Schneider, Walter, Training high-performance skills: Fallacies and guidelines, *Human Factors* **27** (3), 1985, 285-300.

30. Stevens, S.S., On the theory of scales of measurement, *Science* **103** (2684), June 1946, 677-680.

31. Teghtsoonian, J., The judgement of size, *American Journal of Psychology* **78**, 392-402.

32. Tufte, Edward R., *The Visual Display of Quantitative Information*, Cheshire, Connecticut: Graphics Press, 1983.

33. Ullman, S., Visual routines, *Cognition* **18**, 1984, 97-159.

34. Ware, C., and Beatty, J.C., Using color dimensions to display data dimensions, *Human Factors* **30** (2), 1988, 127-142.

35. Yamada, H., An analysis of the standard English keyboard, Department of Information Science, University of Tokyo, Technical Report 80-11, 1980.

36. Zdybel, F., Greenfield, N.R., Yonke, M.D., and Gibbons, J., An information presentation system, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, August 1981, 978-984.